Expertise
and insight
for the future

Thi Le Thanh

Temperature Sensor with RAIN RFID

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor's Thesis

18 May 2020

Author Title Number of Pages Date	Thi Thanh Le Temperature sensor with RAIN RFID 19 pages + 3 appendices 4th June 2020
Degree	Bachelor of Engineering
Degree Program	Electronics
Professional Major	Telecommunications
Instructors	Jukka Voutilainen, Co-founder of Vigilan Oy Heikki Valmu, Principal Lecturer

This thesis presents RAdio frequency IdentificatioN (RAIN), a wireless technology that uses radio waves to identify, communicate and track objects.

The goal of this thesis work was to introduce to wireless temperature sensor network with RAIN technology. The thesis work also aimed to indicate the capabilities of this wireless system and encourage further research in the field of RFID. Furthermore, more advance developments of this application are suggested for another study for students at the Helsinki Metropolia University of Applied Sciences.

The advantages of the system are calrified along with two key components: The smart sensor built as a micro system and RFID (Radio-frequency Identification) embedded reader module. The main purpose of this thesis is to describe how to combine these two different modules based on the same protocol to utilize their application.

Based on this technology, a new approach to measure temperature wirelessly was developed to tackle barriers of wired sensors.

Keywords	RAIN RFID, smart sensor, wireless network, temperature measurement.

Contents

List of figures and tables:

List of Abbreviations

EPC	Electronic Product Code
FCC	Federal Communication Commission
GPIO	General-purpose Input Output
GS1	A not-for-profit organization that develops and maintains global standards for business communication.
IC	Integrated Circuit
IDE	Integrated Development Environment
ISO	International Standards Organization
PTC	Positive Temperature Coefficient
RAIN	RAdio-frequency IdentificatioN
RFID	Radio-frequency Identification
RO	Read-only
RSSI	Received Signal Strength Index
RTD	Resistance Temperature Detectors
RW	Read/Write
TID	Tag Identification
UART	Universal asynchronous receiver-transmitter

UHF	Ultra High Frequency
USB	Universal Serial Bus
USB-to-TTL	Universal Serial Bus to Trasistor-Transistor Logic
WMO	World Meteorological Organization

1 Introduction

Temperature measurement is among the most basic requirements in various fields, for example, construction, agriculture, automation, etc. Currently, there are different available technologies for measuring temperature in the market. However, there is still demand for a new technology that provides a wireless method with a lower cost.

The purpose of this thesis is to introduce the new technology mentioned above. This wireless temperature measuring method is especially useful in situations in which using a wired sensor is impossible. Those situations could be measuring temperature inside building concrete, testing an aircraft engine or other critical structures.

The result of this thesis work is a combination of two different components which are responsible for measuring temperature wirelessly. Generally, those components are a smart temperature sensor and a reader module. This technology can remove the physical links like wires and battery therefore increase the possibility, convenience, safety for any users. The thesis is used as a reference design at Vigilan Oy for future developments.

The first two chapters will present the background knowledge: Chapter 2 will cover the definition of temperature and methods of measuring it. The RFID (Radio-frequency Identification) technology will be explained in chapter 3. Chapter 4 will clarify more technical details about three key components of the measuring system. Chapter 5 will go through hardware and software implementation. And the conclusion will summarize all information including some discussion and suggestions for future research and development.

2 Temperature

2.1 General

2.1.1 Definition

Generally, temperature is the thermal energy presentation of an object and its value is decided by the net flow direction of the of thermal energy between two objects. In a system, the object which has a higher temperature loses heat to the other body.

> Temperature is a physical quantity characterizing the mean random motion of molecules in a physical body. Temperature is characterized by the behavior whereby two bodies in thermal contact tend to an equal temperature. [1]

2.1.2 Units and Scales

The basic temperature is thermodynamic temperature *(T)*, with units of kelvin *(K)*, "Kelvin temperature". The kelvin is the fraction 1/273.16 of the thermodynamic state of the triple states of water, which is solid state, liquid state, and gaseous state coexist in equilibrium. The temperature *(t),* in degrees Celsius, "Celsius temperature" determined by the equation is the most common use in practice. [2]

$$t = T - 273.15 \qquad (1)$$

2.2 Measurement Methods

To measure the temperature of an object by using the heat transfer principal, a thermometer can be brought to the same temperature as the body and then measuring the temperature of the thermometer itself. Otherwise, the temperature can be measured by a radiometer without the requirement of thermal equality.[2]

Temperature measurement methods can be divided into two categories. They are Invasive thermometry and Non-invasive thermometry. Nowaday non-invasive thermometric methods are more desired and widspread than invasive thermometry due to its natural.

2.2.1 Resistance Temperature Detectors (RTD)

A temperature sensor resistance changes with environmental temperature. It is usually made of a platinum wire wrapped around by ceramic robbing. [3, 3.45.] It is widely used because of having great stability, accuracy and repeatability. It also has a good linearity with temperature - the higher temperature, the larger resistance. However, an RTD requires a small constant current to pass through to produce an output voltage, which may cause self-heating.

2.2.2 Positive Temperature Coefficient (PTC)

The concept of Temperature coefficient of resistance is a parameter which shows the relationship between the variation of resistance and the increase in temperature. Positive temperature coefficient means the resistance increases with increasing temperature. [4,291.] PTC thermistors are mainly designed for protection circuits such as transformers and motors. The resistance of PTC thermistors is low and comparatively constant, with low temperature. At a reference temperature, the increase of resistance with temperature becomes very rapid. PTC thermistors are manufactured from compounds of barium, lead, and strontium titanate.

2.2.3 Mercury Thermometer

The meter consists of a bulb containing mercury attached to a glass cylinder of narrow diameter, the volume of mercury in the cylinder is much less than the volume in the bulb. Temperature variation causes the volume of mercury to change slightly. The rest of the cylinder may be filled with nitrogen gas or it may be at less than atmospheric pressure. Because of environmental concerns, the mercury thermometer is being replaced by electronics thermometers. [5, 9-18.]

2.2.4 Radiation Thermometer

Radiation thermometers or pyrometers use the electromagnetic radiated waves from a object to measure its temperature. The total radiation object emit increases rapidly as it warms up and the spectral distribution shifts to shorter wavelengths. Thus by measuring

the radition, the temperature can be determined. [6,3/38.] The clear advantage is the measuring can conduct remotely from the hot body.

3 Radio-frequency Identification (RFID)

3.1 Background Knowledge

RFID is a wireless radio communication technology which is used to uniquely identify tagged bodies. An RFID system combines three basic components as shown in figure 1:

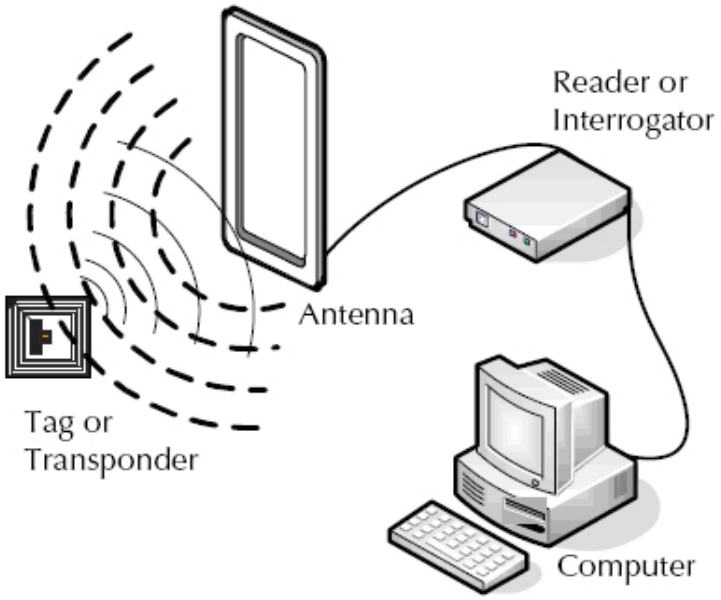


Figure 1: RFID Basic Components (www.epc-rfid-info/rfid)

- A tag or so called a transponder, which is built of a semiconductor chip, an antenna, and sometimes a battery.
- A reader called an interrogator or a read/write device, which is built of an antenna, an RF electronics module and a control electronics module.
- A controller or a computer, which often is a microcontroller running database and manipulate software or so called middleware.

[7, 5.]

3.2 RFID Tag

The RFID tag basic function is to store data and send it to the reader. An RFID tag example is demonstrated in figure 2:

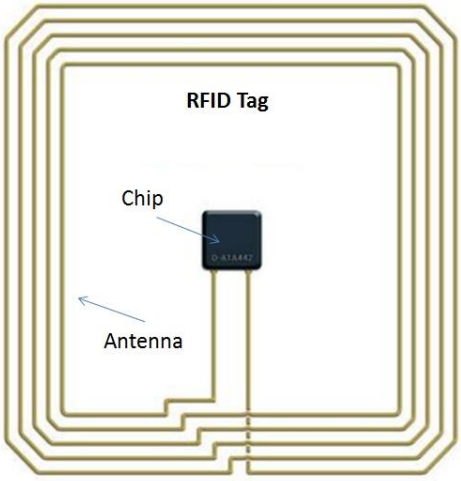


Figure 2: RFID tag

An electronics chip and an antenna are the most basic elements of a tag. Generally, data may be stored and read or write in a chip. Some tags may contain batteries and this is the difference between active tags and passive tags. [7,6.]

3.2.1 Active and Passive Tags

RFID active tags contain an on-board (usually) battery as their power source. When the tags need to send data to the reader, it uses this source to produce the power for the transmission. Relying on this, active tags can communicate with less powerful readers and can transfer data over much longer ranges. In addition, they have larger memory capacity. However, these tags are much larger, more expensive and complex to produce.[7,7.]

On the contrary, passive RFID tags have no on-board power source. As a substitute, they get power to send data from the reader sent signal. Therefore, passive tag is typically smaller and less expensive to manufacture than active tag. However, the active range of passive tag is shorter than the range of active tag. Furthermore, it needs more

powerful reader and has less memory space. Some passive tags have power sources but instead of using these sources to aid in transmitting radio signals, it is used to power other on-board electronics, such as a sensor.

3.2.2 Read/Write, “smart” Tags and Read-Only Tags

Another seperating point between tags is their memory types. There are two types: read/write (RW) and read-only (RO):[7,8.]

- RW tags are often called “smart” tags. They can store a data and have an changeable addressable memory. User can erase and rewrite the data on an RW tag. Nowadays, due to the advantage of smart tag technology the prodcution costs of a tag can be under 1€.
- RO memory is just like its name. The tag memory can be read only. RO tag is programmed once by its manufacturer and unchangeable. This tag also has a limited memory capacity.

3.3 RFID Reader

The RFID reader is the “soul” of the RFID system. It is responsible for communicating with the tags by sending and collecting data as RF waves. The RFID reader also lay out an interface for RFID middleware, which access the tag data. A reader mainly contains a microprocessor and an antenna.

RFID readers usually have one antenna for sending and collecting RF signals. The antenna can come in different forms, such as Vertical, Dipole, Yagi or Parabolic etc. The antenna also is tuned depending on the environment where it will be deployed in. Like any RF antenna, the key performance of it is polarization, bandwidth and gain. It is commonly of reader antennas to be circularly polarized for multipath environments. The bandwidth is another major factor of the reader antenna and determines if a reader be allowed to use globally due to the different area having different permitted different frequency. Gain decides the effective detecting range of a reader, typically the higher the gain the longer the reading range.

The reader's microprocessor processes the information to communcate with the tag. When there is more than one tag present in the reading zone, the reader uses an embedded algorithm to solve the anti-collision communication. The anti-collision must be done on the reader's side and is one of the major functions of a microprocessor. [8,34.]

3.4 RFID Controllers

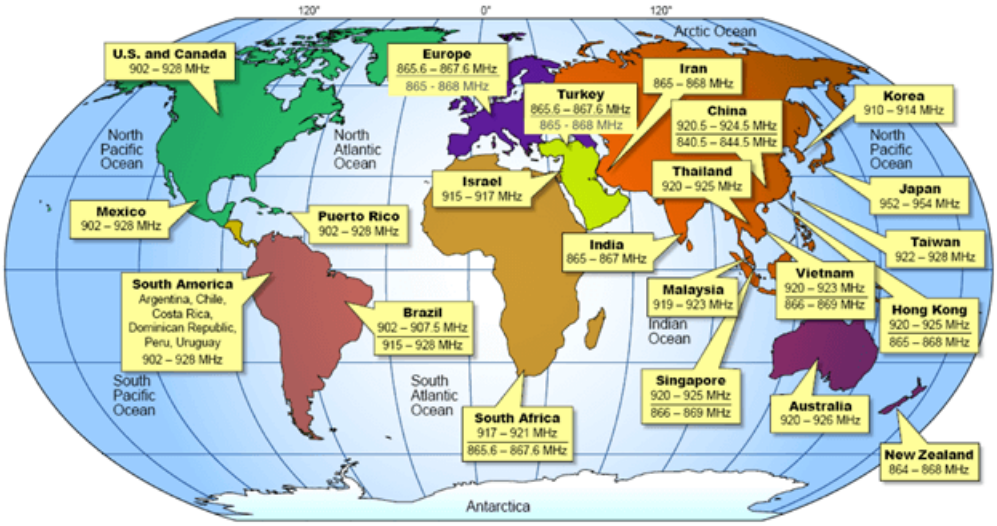
RFID controllers are the "brains" of any RFID system. The RFID controller in any network could be a workstation or Programmable Logic Controller running database. Nowadays, controllers are usually microcontrollers which could use information gathered in the field by the readers. In some case controllers even have I/O pins which can be used to make local options.

3.5 RFID Middleware

RFID middleware is a software placed between the reader and the application. RFID middleware aggregates, filters, manages, formats and converts the data coming from the tags into sensable information. So that a software application can process this data. [8,35.]

3.6 RFID Standard EPC Gen-2

There are many standards managing the RFID systems in RFID field. The thesis will only focus on the Electronics Product Code (EPC) Gen-2 standard. The EPC Gen-2 standard was accepted by International Standards Organization (ISO) as ISO 18000-6C. It is a dominant standard for passive UHF (ultrahigh frequency) RFID system which has frequency band varies from 860 to 960 MHz basing on the geographic area. [9,99.] For instance, in the United States, the frequency range accepted by the Federal Communication Commision (FCC) is between 902 and 928 MHz while the permitted range in Europe is from 865 to 868 MHz. Despite the frequency range the Gen-2 standard establishes the communication link between the tag and transponder.



Firgure 3: Worldwide RFID UHF Map (www.rfid4u.com)

Gen-2 is also the dominant RFID standard in the retail sector. This sector needs an individual ID. The ID will be used as a key to access to collected database. This individual ID is the EPC number whose primary goal is to supply a global system. In this system an item can be individual identified.

The most used features of the Gen-2 protocol are the inventory commands which consist of four commands: (1) Select; (2) Query; (3) QueryRep; and (4) QueryAdjust. Within a range of the reader, the tags EPC numbers can be read using these commands. These readed numbers can be used to access to a database to get more tag information. [9,101.] The second most used feature of the Gen-2 protocol is the write/read commands. Write command allows the user to write one data word,16-bits to a given tag memory location. Likewise Read command enables the user to read up to 256 words.

4 Measuring System Key Components

4.1 Temperature sensor/Smartrac Dogbone

Smartrac is the first RFID sensor using Axzon Magnus S3 which employs Smart Passive Sensing technology to enable a new class of sensors. This new class of sensors does not require maintenance and battery, thus, they are used as replacements for the high price traditional sensors. With the passive nature of the smartrac, the sensor can be

deployed in environments where an electrical source is not possible. The sensor is under governing of standard EPC UHF protocols /EPC Class 1 Gen 2, ISO 18000-6C.

Figure 4: Smartrac Dogbone

As demonstrated in figure 4, the chip of the Smartrac Dogbone has an integrated circuit which can detect silicon die temperature. Due to heat transfer principles, the detected temperature come close to the surrounding temperature materials . The die temperature sensing is an internal measuring IC (integrated circuit) chip temperature.[10,4.] Sometimes, it is preferred as Junction temperature which is the operating temperature of the actual semiconductor in an electronic device.

Table 1: IC Techinical Specifications

Feature	Magnus S3
EPC Memory	128 bits
User Memory	176 bits
(Tag Identification) TID Memory	64 bits
Sensor Code Resolution	512 steps (9 bits)
On-chip RSSI Resolution	32 steps (5 bits)
IC Operating Temperature	-40 *C to 85*C
IC Sensitivity	Read: -16.6 dBm Write: -9.9 dBm
Temperature Code Resolution	4096 steps (12 bits)
Temperature Accuracy	0 *C to 50 *C: +/- 0.3 *C -40 *C to 85 *C: +/- 1.0*C

Beside the capability of temperature sensing, Smartrac Dogbone has two more primary sensing functionalities: Chameleon Engine and On-Chip RSSI (Received Signal Strength Index). With any typical RFID tag, when it is in the presence of detuning

materials such as water or metallic objects, performance loss and frequency shift will occur. The chip with Chameleon Engine can adapt its internal variable capacitance to meet the impedance antenna in the presence of detuning materials. Therefore, it has a self-tuning capability to improve performance. It also can measure the power amount received and sends back this data to the reader. [10,4.]

Table 2: Chip Memory Map

Bank #	Bank Name	R/W	Bit Address	Description LSB MSB	Default Value
11	USER	READ ONLY	E0–EF	Temperature Sensing Enable	N/A
			D0–DF	RSSI Threshold	N/A
		READ/WRITE	B0–BF	Temperature Calibration Data	N/A
			A0–AF	Temperature Calibration Data	N/A
			90–9F	Temperature Calibration Data	N/A
			80–8F	Temperature Calibration Data	N/A
			70–7F		0
			60–6F		0
			50–5F		0
			40–4F		0
			30–3F		0
			20–2F		0
			10–1F		0
			00–0F		0
10	TID	READ ONLY	50–5F	TID[15:0]	
			40–4F	TID[31:16]	
			30–3F	TID[47:32]	
			20–2F	Extended TID Header	
			10–1F	Tag Model Number	
			08–13	Manufacturer ID	
			00–07	Class ID	
01	EPC	READ/WRITE	90–9F	EPC#[15:0]	0
			80–8F	EPC#[31:16]	0
			70–7F	EPC#[47:32]	0
			60–6F	EPC#[63:48]	0
			50–5F	EPC#[79:64]	0
			40–4F	EPC#[95:80]	0
			30–3F	EPC#[111:96]	0
			20–2F	EPC#[127:112]	0
			10–1F	StoredPC[15:0]	0
			00–0F	StoredCRC[15:0]	0

00	RESERVED	READ ONLY	E0–EF	TEMPERATURE CODE	
			D0–DF	RSSI CODE	
			C0–CF	SENSOR CODE	
		READ/WRITE	30–3F	Reserved for future use	
			10–1F	Kill Password[15:0]	
			00–0F	Kill Password[31:16]	

The tag memory storage is divided into four segments. The chip memory map is shown from the table 2. The memory bank number is in binary. The first segment (00) is named the Reserved Memory bank which stores the access password andkill password. The second segment (01) is the EPC Memory bank which contains EPC number located from 20 to 9F bit address .The third segment is the TID or Transponder ID Memory bank which identify the chip's manufacturer and model and the last segment (11) is the User Memory bank where user can read or write their data from 00-7F bit address . [11,3.]

4.2 Reader module/ Nordic ID module

ENUR-05WL2 is a PCB embedded module with "the heart" is NUR-05WL2, which is a compact UHF RFID reader SMD module. It is compatible with EPC Class 2 Gen 2 (ISO18000-6C), Dense Reader Mode requirements; fulfills ETSI, FCC and IC radio regulation. Maximum output power is +27dBm and can be adjusted with 1 dB steps. This embedded module has a capability of reading 200 tags per second and requires 3.6 VDC supply voltage (29W).[12] It also has four antenna connector micro-miniature coaxial (MMX)l types. Figure 4 illustrates an example of a Nordic ID module:

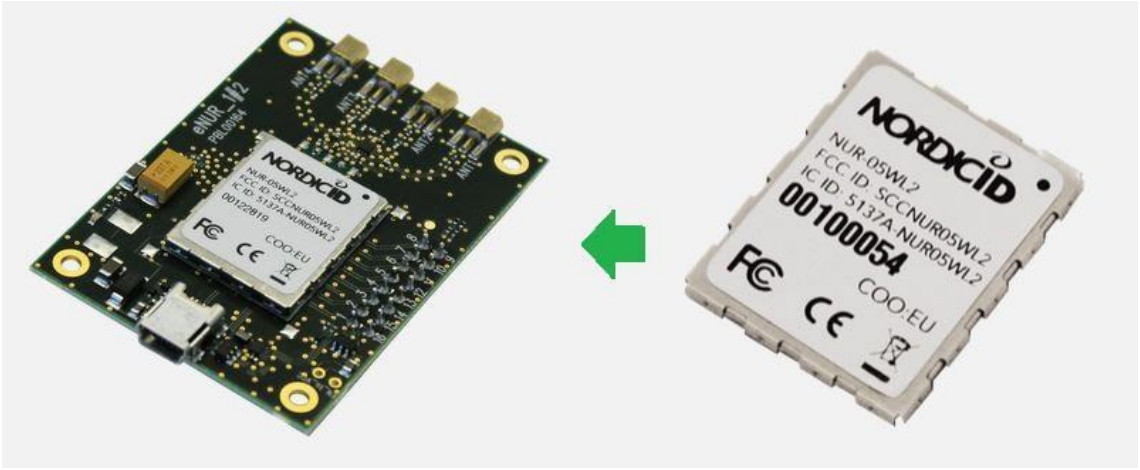


Figure 5: NordicID NUR Modules

NUR-05WL2 has a 3.3V linear regulator and can take input voltage from 3.4V to 5.5V. But it does not have a protecting circuit and being an Electrostatic discharge sensitive component so it must be handled with care. It has five programmable GPIO (General-purpose input output) and can be communicated by UART (Universal asynchronous receiver-transmitter) and USB 2.0 (Universal Serial Bus).

An NUR-05WL2 Block Diagram is illustrated as figure 6:

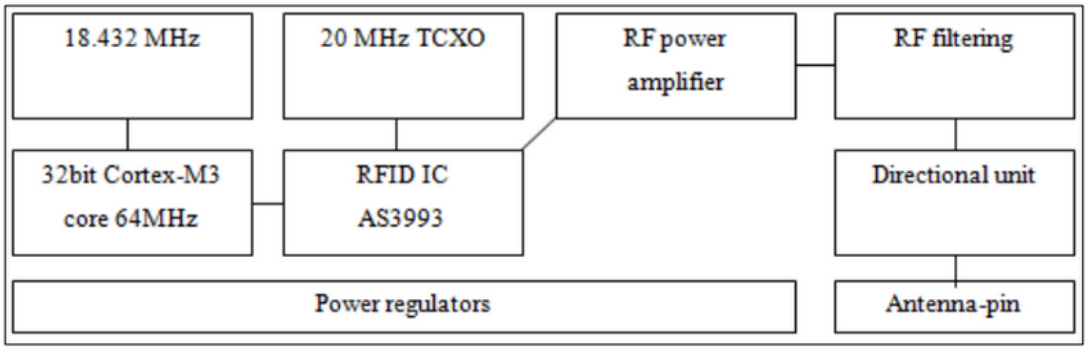


Figure 6: NUR-05WL2 Block Diagram

Other key features of the module are selectable RF parameters:

- Modulation can use ASK (amplitude shift keying) or PR-ASK (phase reversed amplitude shift keying) modulation. The PR-ASK modulation can transfer energy more efficiently to the tag because the RF envelope is higher than it is using ASK modulation.

- RX encoding/ Miller encoding and link frequency affect tag-to-read data rate. The Miller encoding value affects the number of clock cycles that tag uses to modulate one symbol. Therefore, when using higher Miller encoding schemes tag to reader, data will be slower but at the same, it is more robust to interferences. Also, tags response spectrum is denser around the link frequency when using higher Miller schemes. This allows the receiver to use narrower channel filters. Selectable values are M2, M4, M8 or FM0.

- The link frequency affects the frequency offset of tag replying to respect of the reader's carrier wave. For example, when used link frequency is 256 kHz, tag will

reply at the frequency of reader transmission frequency +/-256 kHz. The selectable parameters are 160 kHz, 256 kHz and 320 kHz.

[12]

Table 3: Different Data Rates

Link frequency (kHz)	RX encoding	Tag to reader data rate (kbps)
160	FM0	160
160	M2	80
160	M4	40
160	M8	20
256	FM0	256
256	M2	128
256	M4	64
256	M8	32
320	FM0	320
320	M2	160
320	M4	80
320	M8	40

Globally, the regulations vary depending on the country or part of the world. The below table shows the available options for different regions and the respective frequency band allowed.

Table 4: Pre-programmed countries/regions

Number	Country / region	Frequency / channel BW
0	ETSI / Europe	865.6 – 867.6 MHz / 200kHz
1	FCC / North-America	902 – 928 MHz / 500 kHz
2	People's Republic of China	920.5 – 924.5 MHz / 250 kHz
3	Malaysia	919 – 923 MHz / 500 kHz
4	Brazil	915 – 928 MHz / 500 kHz
5	Australia	920 – 926 MHz / 500 kHz
6	New Zealand	921.5 – 928 MHz / 500 kHz
7	Japan 250mW LBT	916.8 – 923.4 MHz / 200 kHz
8	Japan	916.8 – 920.4 MHz / 200 kHz
9	Custom	840 – 960 MHz

4.3 RAdio-frequency IdentificatioN (RAIN)

RFID encompasses a lot of different types of radio frequency identification technology. In 2014, an alliance was introduced to the market which focuses on a RFID, the passive UHF RFID or RAIN. It has been internally standardized on ISO 18000-63 standard from ISO and Gen 2 UHF standard from GS1. There are many aspects of the standards which define:

- How to create numbering systems.
- How to handle the tag's data once the reader gets a collection of ones and zeros.
- How it is formatted as well as transferred.

The key of RAIN standard is the air interface. It defines how the RFID tags talk to the reader with full encryption techniques employed in the chip itself. This means that the chip and the reader can exchange tokens backwards and forwards to identify whether the reader has the rights to talk to that chip.

[13]

5 Implementation of the system

5.1 Hardware

The hardware is a simple system. Its components include a main power source, voltage regulator, microcontroller and RFID reader. Main power was A 3.7V lithium ion battery which was used to supply enough electric current for the reader with low noise. The switching regulator, MAX631ACPA gives constant 5V for the microcontroller which is ATMEGA328P.

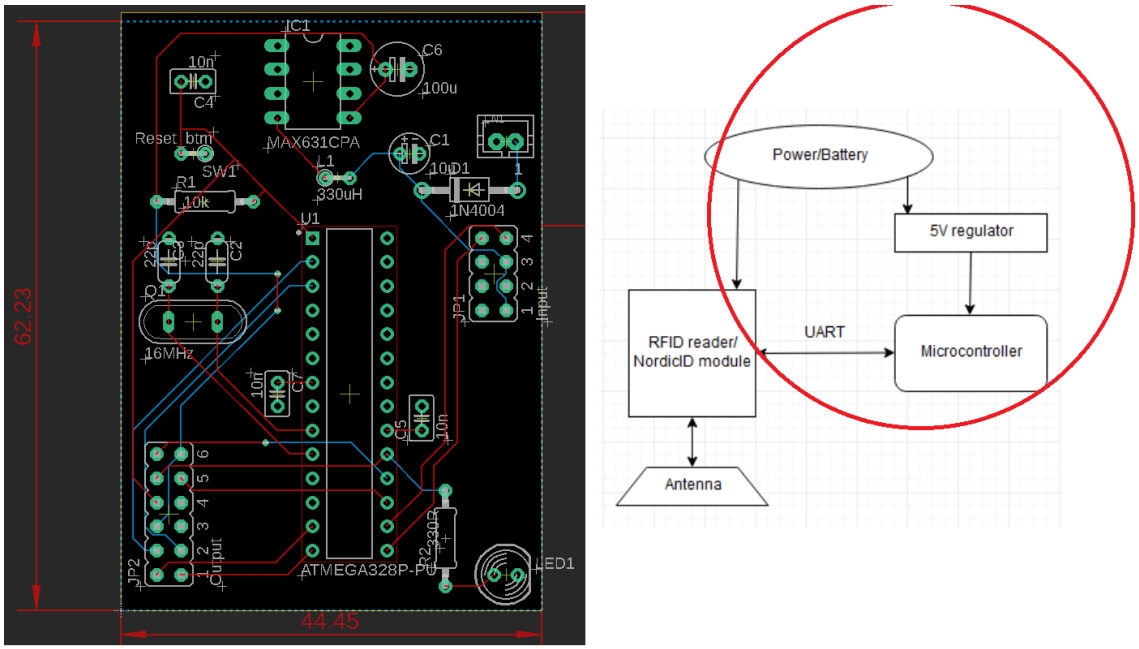


Figure 7: System Diagram

The diagram on figure 5 describes the system components connection. The microcontroller and the reader share the same main power and communicate through an universal interface. Using the Computer-aid Design software Autodesk Eagle, the circuit layout on the right shows the components and the connections in the red circle.

5.2 Software

A simplified explanation on how the reader communicates with the tag by using the EPC Class 1 Gen 2 protocol is the reader will issue an inventory command, then the tag will reply with its EPC. Meanwhile, the tag receives power and continuously writes its sensor value into its own memory. After receiving the tag's EPC, the reader will request data at a sensor memory location. The tag will reply to memory location data containing its sensor value.

Table 5: Temperature Data

Field Name	Memory Bank	Starting Bit No. (MSB)	No. of Bits	Description
CODE1	User (Bank 3_h)	90_h	12	Temperature Code of 1st Calibration Point
TEMP1	User (Bank 3_h)	$9C_h$	11	(Temperature of 1st Calibration Point in ℃) x 10 + 800
CODE2	User (Bank 3_h)	$A7_h$	12	Temperature Code of 2nd Calibration Point
TEMP2	User (Bank 3_h)	$B3_h$	11	(Temperature of 2nd Calibration Point in ℃) x 10 + 800
C	Reserved (Bank 0_h)	E_h	12	Measured Temperature Code to be converted (note: Select command and 3ms continuous wave must be applied before Temperature Code can be retrieved)

The program was written by using the Arduino IDE (Integrated Development Environment) with C/C++ programming language. It issues two inventory commands. The first commands will wake up the tag and read the ID and temperature calibration codes. The second command will read the temperature code. From the temperature code and temperature calibration codes, it will calculate the temperature value by following the conversion formula below:

$$Temperature\ (*C) = \frac{1}{10}\left[\frac{TEMP2-TEMP1}{CODE2-CODE1}(C - CODE1) + TEMP1 - 800\right] \quad (2)$$

5.3 Test Procedure

The main purpose of the test was to evaluate whether the given system performs temperature measurement correctly . To archive this purpose, the sensor tag was placed in different positions near the reader antenna. The reading results could be monitored through a PC computer by connecting a USB-to-TTL (Universal Serial Bus to Trasistor-Transistor Logic) Serial module to the microcontroller serial port.

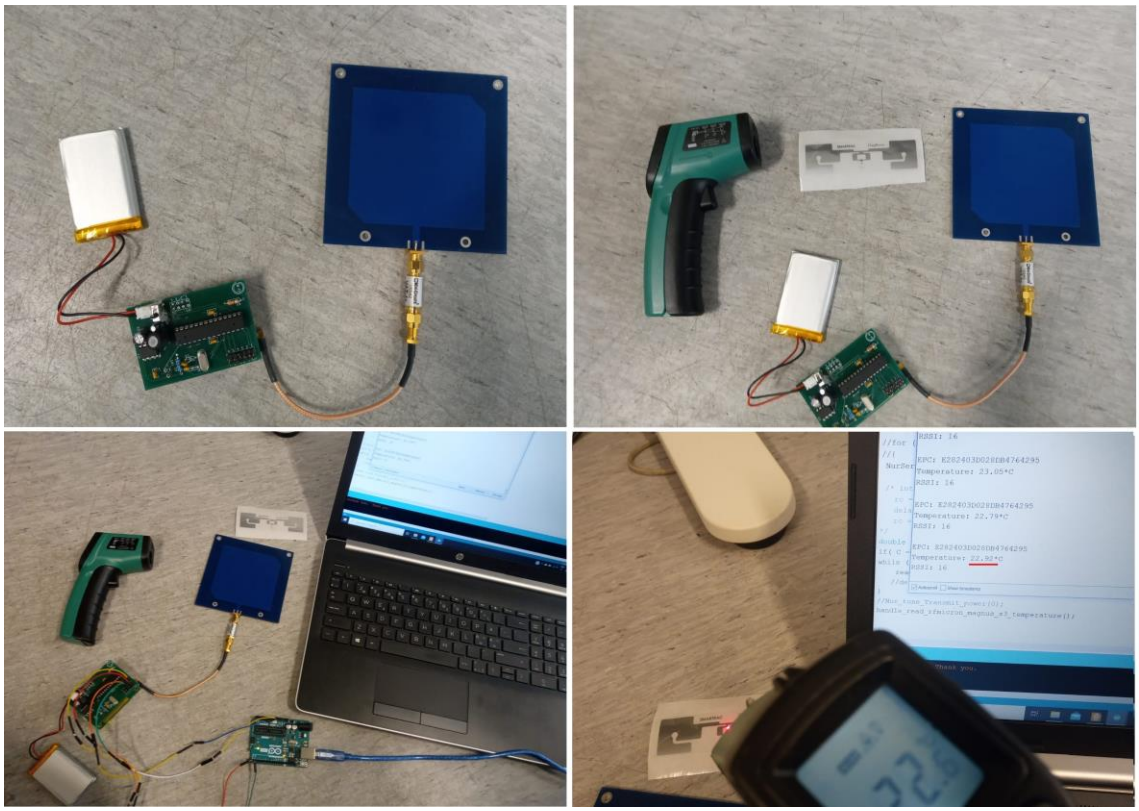


Figure 7: Testing Procedure

The temperature value could be compared with the value of an Infrared Thermometer. During the procedure, the temperature measurement generated by the Infrared Thermometer would be used as the standard result. The accceptable values generated by the wireless system should has the accuraccy of +/- 0.3 Celcius in the range from 0 to +50 Celsius compared to the result measured by the thermometer.

6 Result and Conclusion

The result of temperature measurment is demonstated in firgure 7. The computer screen in firgure 7 shows the values of the measured temperature as well as the sensor ID. As shown in this figure, the difference between temperature mesured by the wireless sensor system and the temperature measured by Infrared Thermometer is 0,3 (°C). This is the expected result.

However, the current system still has inevitable limitations would require further developments. The first limitation is matching the polarization of the tag antenna and the

reader antenna. The second one is the wireless system can not measure multiple tags at the same time. Moreoever, the write feature of the tag has not been ultilized yet. These disadvantages can be improved in the future. Another improvement can be implemented is connecting the system to a cloud server where the measuring result was saved and utilized.

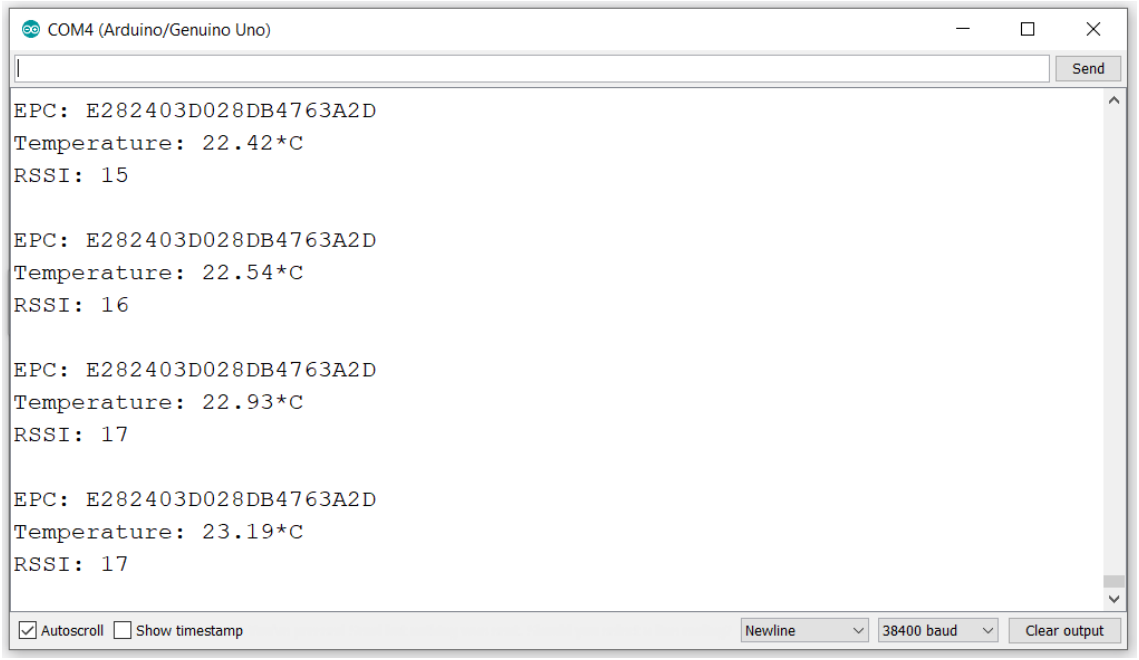


Figure 8: Measurment Result

By using RAIN RFID to read the temperature and ID, the system successfully discards the needs for battery, maintaining cost and wire between the sensor and the reader.

To sum up, the temperature mesurement using RAIN RFID is performing correctly and archive the development purposes. The RAIN RFID technology is successful applied to the system. Besides measuring temperature, the system can also identify unique tagged object with the given EPC code. The limitations mentioned above is minor and possible to be enhanced in the future.

References

1. World Meteorological Organization, 1992, Measurement of Temperature and Humidity: Specification, Construction, Properties and Use of the WMO Reference Psychrometer (R.G. Wylie and T.Lalas).Technical Note No.194 (WMO-No.759). Geneva.

2. World Meteorological Organization, 2002a: Measurement of temperature with wind sensors during severe winter conditions (M. Musa, S.Suter, R. Hyvönen, M. Leroy, J. Rast and B. Tammelin). Papers Presented at the WMO Technical Conference on Meteorological and Environmental Instruments and Methods of Observation (TECO-2002). Instruments and Observing Methods Report No.75 (WMO/TD-No.1123). Geneva.

3. Hank Zumbahlen, with the engineering staff of Analog Devices, in Linear Circuit Design Handbook, 2008 - Chapter 3, Section 3.2: Temperature Sensors, page 3.45.

4. C.Hagart-Alexander, in Instrumentation Reference Book (Fourth Edition), 2010 - Chapter 21: Temperature Measurement, page 291.

5. Fahrenheit, a Pioneer of Exact Thermometry. The Proceedings of the 8th International Heat Transfer Conference, San Francisco, 1966, Vol. 1, page 9–18.

6. Charles J. Fraser, ... (Sections 3.5.1–3.5.8), in Mechanical Engineer's Reference Book (Twelfth Edition), 1994 - Chapter 3: Microprocessors, Instrumentation and Control, page 3/38.

7. V.Hunt, A.Prugila, M.Prugila (Wiley, 2007) RFID - A Guide to RF Identification - Chapter 2: An Overview of RFID Technology, page 5-8.

8. Amin Rida, Li Yang, Manos Tentzeris - RFID-Enabled Sensor Design and Applications (Integrated Microsystems) (2010) - Chapter 2: Fundamentals and Operating Principles for RFID, page 34 - 35.

9. Yan Zhang, Laurence T. Yang, Jiming Chen - RFID and Sensor Networks Architectures, Protocols, Security, and Integrations (Wireless Networks and Mobile Communications) (2009) - Chapter 4: EPC Gen-2 Standard for RFID, page 99 – 101.

10. Smartrac-group - Smartrac Passive RFID Sensors Technical Guide, Page 4.

11. Mouser Electronics - SPSXT001 Smart Passive Sensor for Temperature Sensing datasheet, page 3.

12. NordicID – NUR-05WL2 Manual Intruction.

13. EPC Radio-Frequency Identity Protocols Glass-1 Generation-2 UHF RFID Protocol for Communication at 860 MHz – 960 MHz Version 1.2.0

Appendix

Appendix A: These are the intial setup functions.

```
#include <NurMicroApi.h>
#include <SoftwareSerial.h>


// NUR serial port baud rate.
// NOTE: By default NUR module is configured to 115200 baudrate.
//       If using something else baudrate (e.g. 38400) you'll need to
reconfigure NUR for other baudrate using Nordic ID RFID Configurator
app.

#define NUR_SERIAL_BAUDRATE   (38400)
// Print serial port baudrate. If not used, leave undefined
#define PRINT_SERIAL_BAUDRATE (38400)

// Software serial
SoftwareSerial swSerial(10, 9); // RX, TX

// In this setup NUR is connected to software serial and print data in
HW serial
#define NurSerial swSerial
#define PrintSerial Serial

// In this setup NUR is connected to HW serial and print data in
software serial
//#define NurSerial Serial
//#define PrintSerial swSerial

#ifndef PRINT_SERIAL_BAUDRATE
#undef PrintSerial
#endif

// The API's communication buffers. Adjust if needed
static BYTE ApiRxBuffer[256];
static BYTE ApiTxBuffer[128];

// True if NUR module detected in setup()
BOOL NurAvailable = FALSE;
BOOL NurTemp = FALSE;
BOOL TagCalib = FALSE;



// NurMicroApi handle
static struct NUR_API_HANDLE gApi =
{
  NULL, // void *UserData;
  NULL, // TransportReadDataFunction
  NULL, // TransportWriteDataFunction
  NULL, // UnsolEventHandler;

  NULL, // BYTE *TxBuffer;
  0,    // DWORD TxBufferLen;
```

```
  NULL, //BYTE *RxBuffer;
  0,    // DWORD RxBufferLen;
  0,    // DWORD RxBufferUsed;

  0,    // DWORD respLen;
  NULL  // struct NUR_CMD_RESP *resp;
};

// Read buffer from NUR serial
// If no data available, this function should take 500us - 1000us
int nur_serial_read(struct NUR_API_HANDLE *hNurApi, BYTE *buffer,
DWORD bufferLen, DWORD *bytesRead)
{
  DWORD dwRead = 0;
  DWORD retryCount = 200;

  // Wait for data
  while (dwRead == 0 && retryCount-- > 0)
  {
    // Read all data available
    while (NurSerial.available()) {
      buffer[dwRead++] = NurSerial.read();
      if (dwRead == bufferLen)
        break;
    }
  }

  if (dwRead == 0) {
    return NUR_ERROR_TR_TIMEOUT;
  }

  *bytesRead = dwRead;

  return NUR_SUCCESS;
}

// Write buffer to NUR serial
int nur_serial_write(struct NUR_API_HANDLE *hNurApi, BYTE *buffer,
DWORD bufferLen, DWORD *bytesWritten)
{
  DWORD dwWritten = 0;

  while (dwWritten < bufferLen)
  {
    NurSerial.write(buffer[dwWritten++]);
  }

  *bytesWritten = dwWritten;
  return NUR_SUCCESS;
}

// Init Nur api buffers and transport
void nur_init_handle(struct NUR_API_HANDLE *hApi)
{
  // Init RX buffer
  hApi->RxBuffer = ApiRxBuffer;
  hApi->RxBufferLen = sizeof(ApiRxBuffer);
```

```
  // Init TX buffer
  hApi->TxBuffer = ApiTxBuffer;
  hApi->TxBufferLen = sizeof(ApiTxBuffer);

  // Init transport functions
  hApi->TransportReadDataFunction = nur_serial_read;
  hApi->TransportWriteDataFunction = nur_serial_write;


}

// Print NUR module mode (A = app, B = bootloader) and versions
static void nur_print_versions()
{
#ifdef PrintSerial
  struct NUR_CMD_VERSION_RESP *vr;
  int rc = NurApiGetVersions(&gApi);
  vr = &gApi.resp->versions;

  if (rc == NUR_SUCCESS) {
    PrintSerial.print(F("Versions, mode "));
    PrintSerial.print((char)vr->mode);
    PrintSerial.println("");

    PrintSerial.print(F(" - primary   : "));
    PrintSerial.print(vr->vMajor, DEC);
    PrintSerial.print(F("."));
    PrintSerial.print(vr->vMinor, DEC);
    PrintSerial.print(F("-"));
    PrintSerial.print((char)vr->vBuild);
    PrintSerial.println("");

    PrintSerial.print(F(" - secondary : "));
    PrintSerial.print(vr->otherMajor, DEC);
    PrintSerial.print(F("."));
    PrintSerial.print(vr->otherMinor, DEC);
    PrintSerial.print(F("-"));
    PrintSerial.print((char)vr->otherBuild);
    PrintSerial.println("");
  }

  else {
    PrintSerial.print(F("Version error: "));
    PrintSerial.print(rc, DEC);
    PrintSerial.println("");
  }
#endif
}

// Configure NUR module
static void nur_configure_module()
{

  int rc = NUR_SUCCESS;
  struct NUR_CMD_LOADSETUP_PARAMS params;
  // Flag settings that you want to chage
  params.flags = NUR_SETUP_TXLEVEL | NUR_SETUP_ANTMASK |
NUR_SETUP_SELECTEDANT;
  // Set TxLevel to maximum (500mW/1000mW depending from the module)
```

```
  params.txLevel = 0;
  // Enable antenna 0.
  // Use bit operation if you want to enable multiple
  // antennas like antenna 0 and 1 (NUR_ANTENNAMASK_1 |
NUR_ANTENNAMASK_2)
  params.antennaMask =  NUR_ANTENNAMASK_1 | NUR_ANTENNAMASK_2;
  // Set antenna selection to auto mode
  params.selectedAntenna = NUR_ANTENNAID_AUTOSELECT;



#ifdef PrintSerial
  PrintSerial.print(F("Configure NUR module"));
#endif
  // Set new module setti
  rc = NurApiSetModuleSetup(&gApi, &params);
  if (rc == NUR_SUCCESS) {
#ifdef PrintSerial
    PrintSerial.print(F("OK"));
#endif
  }
  else
  {
#ifdef PrintSerial
    PrintSerial.print(F("SetModuleSetup error. Code = "));
    PrintSerial.print(rc, DEC);
    PrintSerial.println("");
#endif
  }
  rc = NurApiGetModuleSetup(&gApi, &params);
//  PrintSerial.print(params.RxSensitivity, DEC);
}

#ifdef PrintSerial
void print_hex(int val) {
  char tmp[3];
  sprintf(tmp, "%02X", val);
  PrintSerial.print(tmp);
}
#endif
```

Appendix B: When the program calls out this function. The reader will issue an inventory command and return four calibration codes.

```
static int nur_fetch_tags_function(struct NUR_API_HANDLE *hNurApi,
struct NUR_IDBUFFER_ENTRY *tag)
{
  unsigned int aa, bb, cc, dd, ee, ff ;
  int n, i;

 double temp1, temp2 ,code1, code2;
#ifdef PrintSerial
  NumEPC = tag->epcLen;



  aa = (unsigned int)((tag->epcData[tag->epcLen + 0] << 0)); //
Reverse bytes
  bb = (unsigned int)((tag->epcData[tag->epcLen + 1] << 0)); //
Reverse bytes
  cc = (unsigned int)((tag->epcData[tag->epcLen + 2] << 0)); //
Reverse bytes
  dd = (unsigned int)((tag->epcData[tag->epcLen + 3] << 0)); //
Reverse bytes
  ee = (unsigned int)((tag->epcData[tag->epcLen + 4] << 0)); //
Reverse bytes
  ff = (unsigned int)((tag->epcData[tag->epcLen + 5] << 0)); //
Reverse bytes


  code1 = decode_function(aa, bb, cc, 1, 12);
  temp1 = decode_function(bb, cc, dd, 5, 15);
  code2 = decode_function(cc, dd, ee, 8, 19);
  temp2 = decode_function(dd, ee, ff, 12, 22);


  PrintSerial.print(F("Code1: "));
  PrintSerial.print(code1);
  PrintSerial.print(F(" Temp1: "));
  PrintSerial.print(temp1);
  PrintSerial.print(F(" Code2: "));
  PrintSerial.print(code2);
  PrintSerial.print(F(" Temp2: "));
  PrintSerial.print(temp2);

  PrintSerial.print(" EPC: ");
  for (n = 0; n < NumEPC; n++) {
    EPC_cali[n] = tag->epcData[n];
    PrintSerial.print(tag->epcData[n]);
  }

  PrintSerial.println("");

#endif
  return NUR_SUCCESS; // non-zero terminates tag buffer parsing
}


// Perform tag inventory
```

```
// 1. Clear tag buffer
// 2. Perform inventory
// 3. Fetch tags
static void Get_calibration_points()
{

  int rc, n, numTagsMem;
  struct NUR_CMD_INVENTORYEX_PARAMS invExP;
  struct NUR_CMD_READ_PARAMS readP;
  struct NUR_CMD_IRCONFIG_PARAMS inventoryReadC;
  DWORD ticksStart, ticksStop;
  PrintSerial.println("");
  PrintSerial.print("* Get the calibration points");
  PrintSerial.println("");

  ticksStart = millis();

  // Clear tag buffer
  rc = NurApiClearTags(&gApi);



  // Perform EPC inventory + DATA read
  rc = NurApiInventoryEx(&gApi, &invExP);



  //Fetch tags one by one
  numTagsMem = gApi.resp->inventory.numTagsMem;
  NumTags = numTagsMem;

     if (numTagsMem != 0){
    TagCalib = TRUE;
  }

int i;


  PrintSerial.println("");
  for (n = 0; n < numTagsMem ; n++)
  {
    rc = NurApiFetchTagAt(&gApi, TRUE, n, nur_fetch_tags_function);
   (ptr+n)->buffp1 = buffp1;
   (ptr+n)->NumEPC = NumEPC;
   (ptr+n)->buffp2 = buffp2;

      for (i = 0; i < (ptr+n)->NumEPC; i++) {
        (ptr+n)->EPC[i]=EPC_cali[i];
       }

    if (rc != NUR_SUCCESS) {

      break;
    }
  }



  ticksStop = millis();
```

```
    PrintSerial.print(numTagsMem);
    PrintSerial.print(F(" tags in "));
    PrintSerial.print(ticksStop - ticksStart);
    PrintSerial.print(F("ms "));
    PrintSerial.println("");


    // Turn off InventoryRead configuration
    inventoryReadC.active = 0;
    rc = NurApiSetInventoryReadConfig(&gApi, &inventoryReadC);


}
```

Appendix C: Like the first inventory command, this function will command the RFID NordicID reader to issue the second inventory command and return a temperature code.

```
int FetchRfMicronMagnusS3TagsFunction(struct NUR_API_HANDLE *hNurApi,
struct NUR_IDBUFFER_ENTRY *tag)
{
  int n,i;
  double sensorCode;
  double rssiCode;

#ifdef PrintSerial

  PrintSerial.print(F("Antenna "));
  PrintSerial.print(tag->antennaId, DEC);
  PrintSerial.print(F(" RSSI "));
  PrintSerial.print(tag->rssi, DEC);
  PrintSerial.print(F(" ("));
  PrintSerial.print(tag->scaledRssi, DEC);
  PrintSerial.print(F("%) EPC: "));


   rssi =tag->scaledRssi;


  sensorCode = (double)(tag->epcData[tag->epcLen + 1] | (tag-
>epcData[tag->epcLen + 0] << 8)); // Reverse bytes
  rssiCode = (double)(tag->epcData[tag->epcLen + 3] | (tag-
>epcData[tag->epcLen + 2] << 8)); // Reverse bytes
  tempCode = (double)(tag->epcData[tag->epcLen + 5] | (tag-
>epcData[tag->epcLen + 4] << 8)); // Reverse bytes
/*
  PrintSerial.print(F("sensorCode "));
  PrintSerial.print(sensorCode, 1);
  PrintSerial.print(F(" rssiCode "));
  PrintSerial.print(rssiCode, 1);
  PrintSerial.print(F(" tempCode "));
  PrintSerial.print(tempCode, 1);
  PrintSerial.println("");

  */
#endif
for (i = 0; i < NumTags; i++) {
  int counter = 0;
  temp = 0;
  for (n = 0; n < tag->epcLen; n++) {
    if((ptr+i)->EPC[n] != tag->epcData[n])
    { counter = counter+1;
    }
  }
    if(counter ==0)
    {
       temp = ((ptr+i)->buffp1*tempCode + (ptr+i)->buffp2)/10;
       (ptr+i)->temp = temp;
       (ptr+i)->rssi = rssi;
      }
}

    // PrintSerial.println(temp);
```

```
    // PrintSerial.println(rssi);

// PrintSerial.println(temp);
  return NUR_SUCCESS; // non-zero terminates tag buffer parsing
}


static void handle_read_rfmicron_magnus_s3_temperature()
{
  int rc, numTagsMem, n;
  struct NUR_CMD_INVENTORYEX_PARAMS invExP;
  struct NUR_CMD_READ_PARAMS readP;
  struct NUR_CMD_IRCONFIG_PARAMS inventoryReadC;
  DWORD ticksStart, ticksStop;

  PrintSerial.println("");
  PrintSerial.print("* Read Temperature from RfMicron's Magnus S3
tags");
  PrintSerial.println("");

  NurTemp = FALSE;
  ticksStart = millis();


  // Configure InventoryExFilter for Temperature read
  invExP.filterCount = 1;
  invExP.filters[0].action = NUR_FACTION_0; // Matching tags: assert
SL or inventoried session flag -> A. Non-matching: deassert SL or
inventoried session flag -> B.
  invExP.filters[0].address = 0xE0;     // Bit address
  invExP.filters[0].bank = NUR_BANK_USER;   // USER bank
  invExP.filters[0].maskbitlen = 0;       // Zero length mask
  invExP.filters[0].target = NUR_SESSION_SL;  // Set SL flag
  invExP.filters[0].truncate = 0;       // Always 0
  // We must keep CW on during temperature read
  rc = NurApiSetExtCarrier(&gApi, TRUE);
  // Send Gen2 Select + Query
  rc = NurApiInventoryEx(&gApi, &invExP);

  // Reading temperature requires 3ms CW before reader issues any
  // further commands
  delay(3);

  // Clear tag buffer
  rc = NurApiClearTags(&gApi);

  // Set InventoryRead configuration
  inventoryReadC.active = 1;
  inventoryReadC.type = NUR_IR_EPCDATA;
  inventoryReadC.bank = NUR_BANK_PASSWD;
  inventoryReadC.wAddress = 0xC;
  inventoryReadC.wLength = 3;
  // SetInventoryReadConfiguration
  rc = NurApiSetInventoryReadConfig(&gApi, &inventoryReadC);

  // Setup InventoryEx params for RfMicron Magnus S3 inventory
  invExP.inventorySelState = NUR_SELSTATE_SL; // Only tags with SL
asserted responds
```

```
  invExP.inventoryTarget = NUR_INVTARGET_A;   // Query tags with
inventoried flag set to A
  invExP.Q = 1;               // Auto Q
  invExP.rounds = 1;              // Auto Rounds
  invExP.session = NUR_SESSION_S0;      // Session 0
  invExP.transitTime = 0;           // No transit time
  invExP.flags = 0;             // No flags, single inventory
  // Setup RfMicron Magnus S3 filter
  invExP.filterCount = 2;
  // Define a filter to activate On-Chip RSSI Calculation
  invExP.filters[0].action = NUR_FACTION_0;
  invExP.filters[0].address = 0xD0;
  invExP.filters[0].bank = NUR_BANK_USER;
  invExP.filters[0].maskbitlen = 1 * 8;
  invExP.filters[0].maskdata[0] = 0x1F;
  invExP.filters[0].target = NUR_SESSION_SL;  // Set SL flag
  invExP.filters[0].truncate = 0;       // Always 0
  // Define a filter for getting responce only from RfMicron Magnus S3
tag
  invExP.filters[1].action = NUR_FACTION_0; // Matching tags: assert
SL or inventoried session flag -> A. Non-matching: deassert SL or
inventoried session flag -> B.
  invExP.filters[1].address = 0;        // Bit address to start of TID
  invExP.filters[1].bank = NUR_BANK_TID;    // TID bank
  invExP.filters[1].maskbitlen = (4 * 8) - 4; // Set bit length
  invExP.filters[1].maskdata[0] = 0xE2;   // Set TID mask
  invExP.filters[1].maskdata[1] = 0x82;
  invExP.filters[1].maskdata[2] = 0x40;
  invExP.filters[1].maskdata[3] = 0x3B;
  invExP.filters[1].target = NUR_SESSION_SL;  // Set SL flag
  invExP.filters[1].truncate = 0;       // Always 0


  // Perform EPC inventory + DATA read
  rc = NurApiInventoryEx(&gApi, &invExP);
  numTagsMem = gApi.resp->inventory.numTagsMem;
  NumTagstemp = numTagsMem;


  // CW off
  rc = NurApiSetExtCarrier(&gApi, FALSE);


int i;

  for (n = 0; n < numTagsMem ; n++)
  {
    rc = NurApiFetchTagAt(&gApi, TRUE, n,
FetchRfMicronMagnusS3TagsFunction);

      if (rc != NUR_SUCCESS) {
      break;
    }
  }


  ticksStop = millis();

  PrintSerial.print(numTagsMem);
```

```
  PrintSerial.print(F(" tags in "));
  PrintSerial.print(ticksStop - ticksStart);
  PrintSerial.print(F("ms "));
  PrintSerial.println("");



    if (numTagsMem != 0){
    NurTemp = TRUE;
  }
  //PrintSerial.println(NurTemp);

  // Turn off InventoryRead configuration
  inventoryReadC.active = 0;
  rc = NurApiSetInventoryReadConfig(&gApi, &inventoryReadC);
}
```